Google Summer Of Code

LabLua

Interrupt-based drivers and libraries for Ceu-Arduino

Milestone Report 1

Student:
Guilherme Simas

Mentor:
Francisco Santanna

May 24, 2017

Contents

1 Introduction

The goal of this report is to show the progress on the project from the past two weeks, comparing it with the proposal in order to avaliate it. In the first section the expectations for the proposed deadline (May 12th) will be listed, followed by what was achieved.

This document concludes by discussing the next steps forward and proposing alterations to the current scheduling and planning, if needed.

2 Milestone Objective

The goal for this milestone, as described in the current schedule, is:

1. Study current implementation of modules in Arduino which freeze the application.
2. Deepen into Ceu's current implementation of drivers and libraries.
3. Deliver a document with proposed modules for which to develop interrupt-based drivers and libraries.

3 Results

Progress on each item in the previous section's list will now be presented below.

3.1 Study current implementation of modules in Arduino which freeze the application

The Arduino programming model us structured, serial code. Fuctionalities are associated to function calls, and it is expected that after the code is past that line of code, that functionality has completed what is expected of it. For that reason, all functionalities that envolve hardware support freeze while waiting for the hardware to complete its work.

Having said that, that are functionalities and modules which make use of interrupts, for example Serial communication interfaces. The modules usually have a sort of **begin()** call, which initializes the routines. Calls to **read()** and **write()** functions still freeze the application though, as they must still wait for the hardware.

Since all modules freeze and can be improved upon, a selection will be made based on which modules are most relevant for general applications.

3.2 Deepen into Ceu's current implementation of drivers and libraries

Drivers and libraries in Ceu-Arduino are implemented using a combination of functions developed in C which are wrapped in Ceu, due to its simple C integration, and ones fully developed in Ceu. The module **timer.ceu** for example, contains more C implementation than Ceu. On the other hand the module **usart.ceu** presents more Ceu code.

3.3 Deliver a document with proposed modules for which to develop interrupt-based drivers and libraries.

These are the chosen modules for reimplementation. They were chosen based on their utility and frequent use for general embedded systems applications. The goal is to be able to develop a robust application fully implemented using Ceu-Arduino and with no computational and energy waste due to polling.

3.3.1 Analog I/O

The current implementation for the Analog functions for Arduino, such as analogRead(), cause the application to freeze due to polling the hardware register bits which signal the end of the conversion. The Analog To Digital Conversion hardware supports interrupts and therefore, this is a viable choice as interrupts can be used to signal the end of the conversion and avoid polling.

3.3.2 SPI

Wired communication protocols such as I2C and SPI are often used in embedded systems as they enable the use of external components. The current wired data transfer implementation of such protocols in Arduino is done using interrupt service routines for data buffering but the operations still freeze to maintain synchorism in the application (it is expected that a read is through and done after a call to analogRead(), for example, since the code is structured sequentially).

The candidates for this section were I2C and SPI. When it came to deciding between both, SPI was chosen over I2C because the latter's advantadge is that it uses less lines and does a better job when supporting multiple masters and multiple slaves, while the former is faster and simpler. Since in most applications these protocols are used for communication over a microcontroller and an external component, there is rarely need for multiple masters.

This decision is also supported by the fact that most external components on the market (more especifically wireless communication modules) use the SPI protocol.

3.3.3 Serial

For similar reasons to the previous section (SPI), the Serial communication module currently implemented in Arduino blocks the application. Since the Serial communication is an integral part of many embedded systems applications, this is a module which will be of utility for the community.

3.3.4 External RTC

An external RTC can enable the microcontroler to go into a deeper state of sleep and therefore, save power. Developing a module which supports such a component will be a feat for this project in terms of energy-saving, as the amount of idle time gained from all the reimplementations can be better taken advantadge of by entering a deeper state of sleep.

3.3.5 EEPROM

The Arduino hardware offers support to EEPROM operations. The EEPROM can be interpreted here as a small hard-disk inside the microcontroller. As such, information is preserved across resets. Being able to store information in such a manner opens the window to many interesting applications.

Support to these memory operations to the microcontrollers EEPROM save cycles because memory access is slower than most operations. Fortunately, the hardware for EEPROM interface supports interrupts and therefore offers a viable implementation path. This is predicted to be the hardest module subject for implementation because there are, inherently, many challenges in dealing with memory (such as concurrent access to data and memory coherence).

4 Conclusion and steps forward

The project is progressing inside what is expected and, therefore, no changes to the proposal are needed. Implementation of the modules will follow a process of studying the current implementation; understanding the blocking process and motives behind it; proposing a solution; implementing the new module in C; implementing the module in Ceu-Arduino; and finally, developing a sample application.